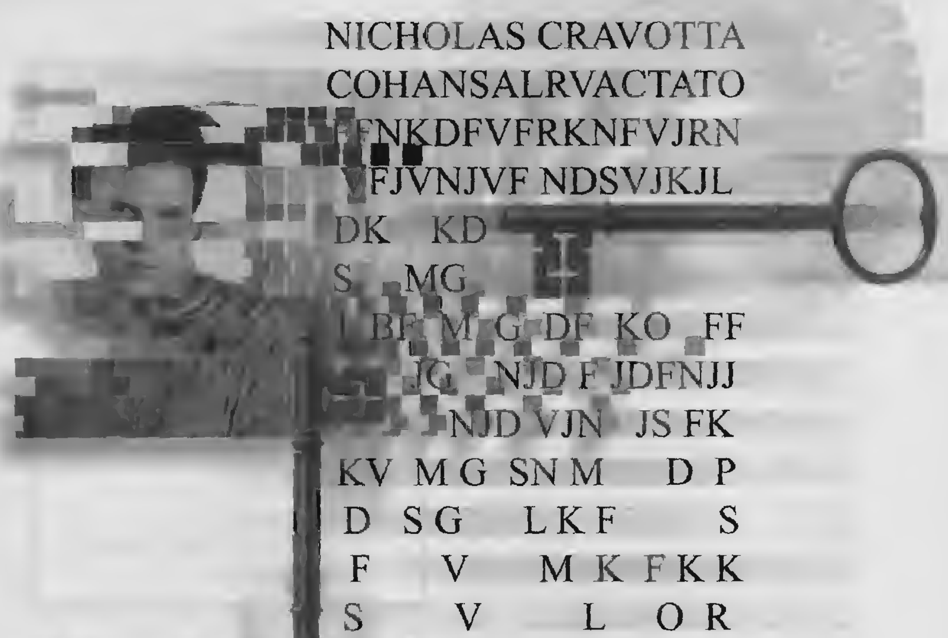
 By Nicholas Cravotta, Technical Editor



EnCRyPtiON: more than just complex algorithms

MUCH OF THE INFORMATION AVAILABLE ON ENCRYPTION FOCUSES ON THE MATH BEHIND TRANSFORMING DATA OR CREATING KEYS. ENCRYPTION, HOWEVER, IS ONLY ONE PART OF A COMPREHENSIVE PROTECTION SCHEME. JUST AS IMPORTANT AS A STRONG ALGORITHM IS A SECURE IMPLEMENTATION OF THAT ALGORITHM.

THE RISE OF DIGITAL TECHNOLOGY HAS CHANGED the way people use and store information. As more and more data takes a digital form—shifting from physical media, such as film, tape, or paper, to bits—the need to protect both the content and the privacy of information has increased. For example, music and video copied from a digital versatile disk (DVD) results in a perfect copy, a factor that has held up the adoption of DVD because studios are afraid to leave their valuable content so vulnerable to theft.

Encryption technology plays an important role in maintaining the general availability of information and controlling its use and abuse. Only those individuals with access to the proper key can read encrypted data. An electronic cracker trying to break an algorithm with a 56-bit key needs to try random keys from a key space of 2^{56} keys. Depending on the speed of the processors that crackers use, cracking can take hours to centuries. In general, the larger the key space, the stronger the algorithm.

The perception that encryption strength depends wholly on the encryption algorithm is false. Given the large key spaces and complexity of today's encryp-

tion algorithms, breaking the algorithm is the last place an attacker attempts to compromise a system. Strong encryption can protect information, but only if the "box" housing the encryption is at least as strong; for example, if an attacker can access information before the system encrypts it, then there is no need to break the encryption algorithm. No product is an island, especially when it comes to security. You may build the part of a system that actually employs encryption technology, but the rest of the system—if system manufacturers don't take care—can nullify your hard work: Security is only as strong as its weakest link. Additionally, the more complex the system, the more potentially vulnerable it is to attack.

You should consider several important factors to successfully implement encryption and reduce system vulnerabilities. These factors include selecting an algorithm; securely implementing encryption; managing keys; and balancing performance, price, and privacy to achieve sufficient security.

With a complete understanding of security and encryption issues, you can begin to compensate for potential vulnerabilities in other parts of a system or point them out to the engineering teams designing these systems. System issues may include guaranteeing the integrity of data from unauthorized tampering, such as insertion, alteration, destruction, and replay. Another issue is guaranteeing confidentiality and privacy. It may be important to protect information flow and content from disclosure. (For example, viewers may want to keep private the kinds of movies they watch.) Guaranteeing authentication is yet another issue: All parties are who they say they are, and only authorized parties have access to information. Repudiation is also a factor. Transactions are immune to false denial (for example, "I never ordered that") through proof of origin and delivery. Finally, consider restoration: A system and its information can survive and resume service after an attack (for example, a set-top prepay procedure is broken).

AT A GLANCE

- ▷Sufficient security is a balance of price, performance, and privacy.
- ▷An application may require several encryption algorithms working together to guarantee data integrity and authenticity.
- ▷Encryption technology is available as software, ICs, or embedded cores.
- ▷Biometrics and tokens, such as smart cards, protect keys from misuse and unauthorized copying.
- ▷The more complex the system, the more vulnerable it is to attack.

NEW STANDARDS FOR OLD

A multitude of encryption algorithms and protocols is available, depending on your application and the required level of security (**Table 1**). For some applications, formal or de facto standards exist, including the Data Encryption Standard (DES) for financial transactions, the secure-sockets layer for networks, and the Internet Protocol Security Protocol (IPSEC) and Secure/Multipurpose Internet Mail Extensions (S/MIME) for the Internet. Several message-authentication, or "hash," standards also exist, such as Secure Hash Algorithm (SHA-1) and message-digest 5 (MD5), which produce a fixed-length value guaranteeing the integrity of a message and which an attacker cannot use to derive the original message. Many of the standards, such as S/MIME, are protocol definitions based on base encryption algorithms, such as DES, Triple-DES, and RC2 (Rivest's Cipher).

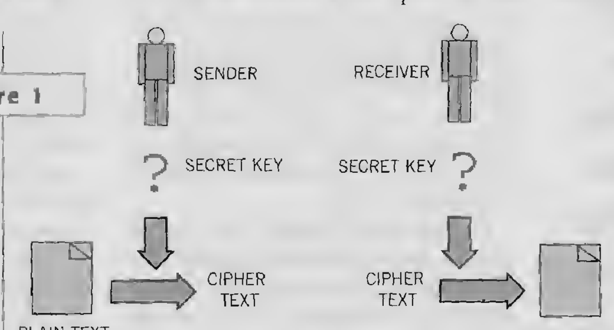
By far, the most widely used algorithm is DES, employing a 56-bit key on a 64-bit data block. It is possible, however, for a cracker to break a DES cipher in less than a few hours (**Reference 1**). The issue is not whether a cracker can break DES 56-bit keys by brute force but how cheaply it can do so.

Industry insiders are already taking steps to replace 56-bit DES with an algorithm using a larger key space. The Advanced Encryption Standard (AES) is the official successor to DES, but it won't be available until 2001. Until then, the National Institute of Standards and Technology (NIST) recommends the use of Triple-DES, or ANSI standard X9.52. Triple-DES churns a message through a DES engine three times. The standard offers several modes supporting three keys per transaction, as opposed to one, and alternates encryption and decryption.

In addition to standard algorithms such as DES, many proprietary schemes offering varying levels of strength, such as elliptic curve and dynamic substitution devices, are available. The difference among these algorithms is their mathematical bases: DES uses S boxes, public-key encryption uses large prime numbers, and several of the next-generation algorithms use modified Feistel networks. Unless you plan to code an algorithm yourself, you need not dive into the details of each algorithm type (**Reference 2**). The key difference in schemes is the varying processor and memory requirements for implementing the scheme. DES has been successful because of the small cryptoengine necessary to manipulate 56-bit keys. A 1024-bit key algorithm offers greater security but requires significantly more processing power. In any case, be wary of vendor claims and benchmarks. For example, you may hear that a 160-bit elliptic key is equivalent to a 1024-bit RSA (Rivest, Shamir, and Adelman) key. Be sure to ask for concrete verification of such claims and ask competitors for their takes on such claims.

The open/proprietary issue takes a different angle with encryption technologies. Certainly, you can prove an algorithm weak by breaking it, but no known means exists for proving an algorithm secure. With the tools for reverse engineering, reliance on the "secret" properties of an algorithm is little security. Thus, there is some-

Figure 1



Using symmetric encryption, both the sender and the receiver use the same secret key. Exchanging the secret key requires a secure channel.

thing to be said for the rationale that an algorithm should be strong enough so that publishing it does not weaken it. In general, confidence in an algorithm increases with time, given that none of the experts of the day can crack it. New algorithms often do not have the scrutiny that old algorithms had.

You can take a hard look at the next-generation algorithms, such as Twofish, RC6, and Cast-256, competing to replace today's "obsolete" standards by reviewing the work that NIST is doing on AES. NIST is running benchmarks for each of the algorithms under various operating conditions, as well as gathering qualitative comments from the encryption community (**Reference 2**). Those algorithms that NIST does not choose to become AES will still be available. For example, if you control both ends of the communication and do not need to interact with any outside network, you have complete freedom to choose the most appropriate algorithm, based on the required level of security, performance, throughput, cost, and size of your pipeline in both directions. You are not bound by NIST's choice of algorithm and can select the best algorithm for your system needs.

In any case, DES will be around for at least as long as legacy systems require compatible support. There is already extensive investment in DES, which will probably for some time color the evaluation of how DES adequately meets needs.

If short keys have "killed" DES, why not just use extremely large keys to begin with and evade the issue of ever cracking an algorithm? The trade-off is balancing performance, price, and privacy. Algorithms such as RSA can employ long 1024-bit keys and processor-intensive algorithms, effectively making the possibility of cracking a message zero. Long keys and complex math, however, though they offer more strength and security, require significant processing time (even minutes), making them infeasible for bulk encryption. One option is to use both kinds of encryption together.

Algorithms such as DES are symmetric, or synchronous, algorithms: Both the sender and the receiver use the same secret key (**Figure 1**). Symmetric algorithms employ transposition and substitution, operations that many processors handle quickly. Asymmetric, or asynchronous, algorithms, such as RSA, on the other hand, use a private key that only the owner of the key knows and a public key that others use to communicate with the owner (**Figure 2**). Because one of the keys is public, there is no need to have a secure channel over which to share the secret key of a symmetric algorithm. (If you have such a secure channel, why not just send the message?) Asymmetric algorithms, such as those based on prime numbers, require multiplication, which tends to significantly slow processing. Asymmetric algorithms also use much longer keys (greater than 400 bits).

A hybrid option uses both kinds of algorithms, employing a "digital signature" (**Figure 3**). Using a symmetric key, the sender encrypts the message more quickly than he could with an asymmetric key. The sender then encrypts the symmetric key using the public key of the receiver. Only the receiver has access to the receiver's private key, which the receiver uses to decrypt the secure symmetric key, which, in turn, the receiver uses to decrypt the message. "Public-key infrastructure" refers to the use of both encryption and digital signatures.

IMPLEMENTATION ISSUES

Encryption is available as an off-the-shelf technology, and vendors often provide it as C code (sometimes as source code) or as a core for or already within ICs. In either case, evaluating encryption implementations can be challenging: The algorithm may be strong, and the demo may run like a champ, but just how secure is the implementation? You can't easily tell by looking at the outside of a chip or at complex code. Attackers are ruthless and attempt to create faults in whatever ways they can. For example, attackers may not answer questions nicely, instead bombarding a system with thousands of answers when the system requires only one and trying to blow a buffer (overflow) or cause a boundary error that the

"RANDOMNESS" IS NEXT TO "SECRETNESS"

Random numbers are critical in generating difficult-to-break keys. Simple pseudorandom-number generators (PRNGs), however, generate repeatable pseudorandom sequences and therefore repeatable sequences of keys. Thus, an attacker with the right initial seed could theoretically generate the same sequence of keys and thus reduce the search space of keys to a trivial size for both past and future messages.

Computers tend to be highly predictable, which makes them useful for computation but poor for generating true random numbers. For example, if the system clock updates only 17 times/sec, knowing the approximate time someone generates a key reduces the unpredictability of seeds based on the system clock. In general, system values are not secure, because an unscrupulous user could access or guess the state of the system at the time of key generation, compromising the anonymity of the key. Even system characteristics that seem random, such as disk access and process and thread events, might have a hardware or software correlation that makes them predictable. A random number must come from a source outside the computer, preferably from several mixed sources that are unavailable to potential attackers. Typical sources include microphone inputs, a user who is randomly striking keys, and temperature variations. Some of these sources, however, may still offer significant predictability, such as a user who repeatedly strikes the same key. The source must also supply enough entropy (unpredictable detail) to meet your volume of keys; generating a 256-bit key with only 160 random bits misses the point.

In addition to beginning with a truly random seed, a secure PRNG should treat every seed bit equally so that no bit affects the output more than any other. The PRNG should also maintain a large internal state, making it difficult to predict or track future states and protect the generator state with the same level of security as a key. If an attacker can compromise and view the internal state of the generator, methods such as additional seeding ensure that the next output is impossible to determine with confidence.

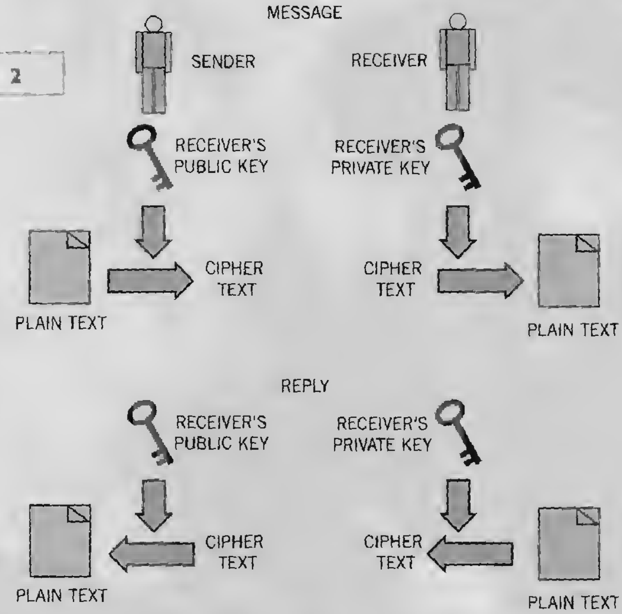
system is unprepared to handle. Attackers search out vulnerabilities, such as areas in which sensitive information resides or how such information is shared. For example, a poor pseudorandom-number generator can unintentionally generate weak keys (see **sidebar** "Randomness" is next to "secretness"). "Invalid" or "misinformed" data can cause errors in trusted code. You can often avoid such errors by clearly defining data-input limits and lengths and by parsing all data through centralized validation checks (versus checks scattered throughout the code).

Another important difference among implementations is the packaging of the algorithm. A development kit for both hardware and software with a functional application-programming interface, clear documentation, and code examples can be worth much more than you pay for them in time-to-market savings. Many encryption companies offer engineering services, acknowledging that there is more to overall system security than the encryption algorithm itself. You can purchase various levels of encryption from base data encryption to "complete" implementations, which include digital signatures, certificate-authentication management, and support for smart cards.

Code and memory size (footprint) require a careful look. Many algorithms take the form of C source code—not assembly—for platform portability. When you evaluate an implementation, try to compile the code using several compilers; the compiler you use can significantly affect encryption speed, based on how it handles shifting operations. Also, consider how efficiently the algorithm is coded. Because the bulk of encryption involves many loops, loop unrolling can significantly increase performance, allowing you to balance the specification of a high-performance processor and more memory.

If you need to interface to other systems and therefore other implementations of the same algorithm, proper implementation is important for preventing interoperability vulnerabilities. You can run your own validation tests, including the public tools manufacturers usually create when the industry adopts a standard. These tools pass test vectors through the implementation and compare the encrypted output with reference vectors. Additionally, remember that, although the algorithms themselves may be free, their implementations aren't. Having to license three algorithms—symmetric, asymmetric, and hash—increases costs, and you may have to license more than one of each to maintain legacy compatibility. Finally, be aware that export laws for encryption are complex and strict (see **sidebar** "Exporting encryption technologies").

Figure 2



Using asymmetric encryption, the sender uses the receiver's public key to encrypt, and the receiver uses a private key to decrypt. Exchanging keys requires no secure channel. This method ensures that a reply is from the receiver because only the receiver has access to the private key.

WHERE DOES ENCRYPTION RESIDE?

Encryption can reside in many areas in a system. In a virtual private network (VPN), for example, encryption can reside in the router chip, on the router board, or at the application level. If you design routers, you may not even need to add encryption yourself; you can bundle someone else's software with your board

EXPORTING ENCRYPTION TECHNOLOGIES

Much misinformation exists regarding US encryption-exportation policy. Previous, "backdoor" policy allowed the export of encryption technologies that support keys as long as 56 bits if the technologies added key-recovery mechanisms. US policy has since removed the backdoor restriction. Currently, US companies can export encryption technologies that support keys as long as 56 bits except to embargoed countries without a license but after a one-time technical review or license exception.

Algorithms that use keys longer than 56 bits require a license, and you can export encryption technologies that support keys as long as 64 bits for certain mass-market products. Foreign subsidiaries and such industries as financial, health, and medical, may be able to export to other countries for their use. This situation has opened the door for US companies to purchase US encryption products and use them in foreign countries, but this situation still does not address the desire of US OEMs to sell to foreign companies. US encryption companies face a challenge: Foreign companies have access to the same encryption ideas and algorithms but aren't bound by US regulations and can sometimes sell to anyone in the world, giving them a competitive advantage. For strong encryption, it may make sense for multinational companies to buy outside the United States. For the latest updates on what's legal, visit the Bureau of the US Department of Commerce's Bureau of Export Administration's Web site at www.bxa.doc.gov/.

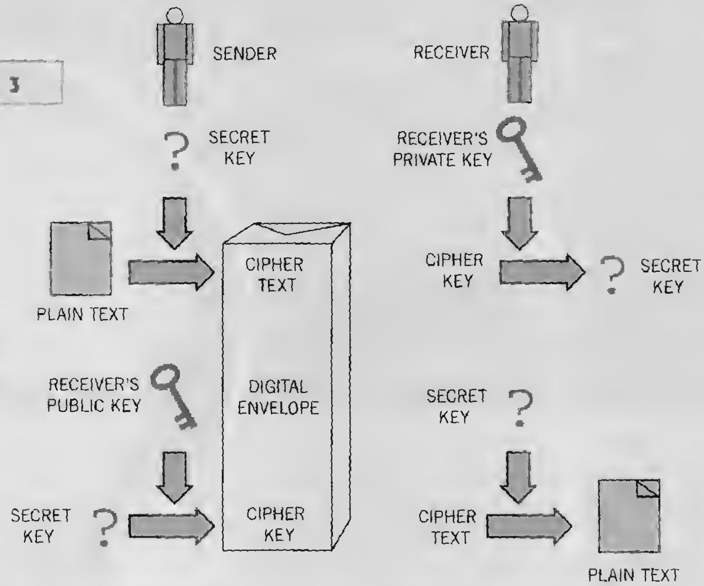
and avoid that part of the design. You can also add encryption as an external box or accelerator card for use with legacy equipment; placed in line before the router, the box transparently encrypts or decrypts information going to or coming from the router.

Employing encryption in hardware offers greater internal security and encryption speeds than software but offers less flexibility for mass redeployment or upgrades. You might consider a programmable approach in which you can later reconfigure the encryption engine, but such a system offers a new vulnerability: An attacker can possibly reprogram the engine and gain access to all messages passing through the engine.

In addition to protecting information that passes through a device, encryption can protect a system from viruses—malicious data intended to crash a system (for example, an invalid MPEG stream). It can also protect against "spoofing" (someone pretending to be a trusted source) and "man-in-the-middle masquerading" (for example, tricking a browser into viewing modified Web pages). Another example is a Java set-top box, which you may configure to allow on-the-fly reprogramming. An attacker could bypass pay-per-view restrictions or infect a network of boxes with a virus by "updating" internal code. By requiring digitally signed and encrypted code updates, you make such an attack more difficult. For third parties writing software for your final platform, digital signatures tell you who sent an update and guarantee that no one has altered the code, but they do not ensure that the code is without errors or that it is not infected with a virus. Your platform must be bulletproof against all code that an attacker could possibly download.

System-bandwidth requirements can affect the placement of encryption. Employing software on a router can significantly reduce the throughput of the router because of the increased processing overhead dedicated to encryption. If you change keys with every transaction, you may slow throughput by being unable to generate a high enough volume of keys fast enough or by taking too long to access keys. If bandwidth is tight, you may want to reduce the size of a key by selecting a different standard, such as a 160-bit elliptic key instead of a 1024-bit RSA key.

Figure 3



Using a digital signature or digital envelope, asymmetric encryption provides a secure channel to exchange a symmetric secret key by encrypting the secret key using the receiver's public key.

CLOAK-AND-DAGGER ATTACKS

The hybrid schemehas several vulnerabilities. For example, the receiver must be able to verify that no one has tampered with the message. Hash functions, such as MD5, provide one-way encryption of a message or the equivalent of a complex checksum. Both sides perform the hash; the sender encrypts the hash value with the symmetric key. And, if the results match, you can confidently as-

OTHER ENCRYPTION TECHNIQUES

Compression: Brute-force attacks assume that the decrypted message is obviously a message; that is, it has some telltale signature or pattern, such as "The secret message is:." Such a pattern may not be uncommon; for example, all credit-card transactions may begin with the same header to indicate the type of transaction. By removing the pattern from the message before the sender encrypts it, an electronic cracker has more difficulty determining that it has found a valid message and thus the right key. Thus, compressing messages and keys in digital envelopes before encryption offers another layer of protection. A similar method, which Triple-DES (Triple-Data Encryption Standard) uses, encrypts the message more than once in succession.

Steganography: Steganography, or data hiding, buries valuable information within less valuable or decoy information, taking advantage of the fact that messages, files, and data often contain unused and padded or insignificant areas. For example, streaming video could contain set-top-box commands invisible to the viewer. One advantage of steganography is that a successful attacker may not know that the data contains a hidden message.

Cognometrics: Password generation has come a long way from using an easily accessible social-security number. Cognometrics offers a creative means for "remembering" a password, such as using pictures instead of alphanumeric characters or asking a battery of preanswered questions to which only one person would know all of the "correct" answers (for example, the question, "What is your favorite beer?" and the answer, "The kind in my refrigerator"). Methods such as cognometrics also prevent attackers from using a password in another, less secure environment.

sume that no one has altered the message.

Another vulnerability is authentication: The receiver needs to know that the other party is indeed the right party and not a spoofer. Verifying the identity of the sender is fairly straightforward: The sender encrypts an identification (ID) message using the sender's private key, and the receiver decrypts it using the sender's public key. (Communicators should use an ID message only once to prevent a spoofer from using it later.) A new problem arises: The public keys must come from a trustworthy source. A hacker can access keys posted on a Web page and replace the receiver's public key with the hacker's own. This problem presents the man-in-the-middle attack, in which a sender encrypts a message using the attacker's public key. The attacker intercepts and decrypts the message and then encrypts it using the receiver's public key and passes the message on to the receiver. The result is that neither party knows that an attacker has intercepted or compromised the message. To answer this need, "certifying authorities" provide certificates containing public (now semiprivate) keys in a secure fashion using the certifying authority's private key. But attackers can break certifying-authority keys, forcing the certifying authority to revoke the certificates.

And so on. Encryption involves a lot of cloak-and-dagger, and attacks can be complex and subtle. For every preventive measure, a corresponding countermeasure exists. How careful your implementation needs to be depends on the calculated risk of data loss. In any case, keeping public keys relatively secret and frequently changing them should be an integral part of a system's design, but don't count on the secrecy to protect them! Attackers can compromise or guess secrets. If you don't employ a third party as a certifying authority, you take on the role yourself and need to manage access to your public keys.

KEY MANAGEMENT

The value of a key is directly proportional to the value of the information it protects. If one key protects an entire system, cracking that one key is worth everything in the system. If one key protects one user, cracking the key is worth the information available to that user. A key that protects only one transaction is worth the least in cost versus value gained and thus offers the best protection. However, using a key requires a rigorous key management, adding complexity to a system and opening another door to attack.

An important foundation for key management is that the cryptoengine automatically generates, exchanges, uses, and discards keys without human intervention. In this way, a human can never access a private key, ensuring that no one can copy, misuse, or give away the key. Tamperproof containers should "zero out" a key if anyone ever compromises the container. There should be no external access to the memory holding a key or to the memory bus. Protecting keys outside a tamperproof container may be more difficult than you think. For example, a key hidden on a network may reside in more than one place: temporarily in a cache, in a recently freed memory allocation (memory-reconstruction attack), or in the discarded shell of a deleted file.

You must also support a mechanism for revoking keys. This feature becomes necessary if someone compromises a key; that is, the key breaks or the person holding the key becomes untrusted (for example, fired). One method is to maintain a list of revoked keys. In a limited-resource environment such as a set-top box, however, you can record only so many revocations before either the buffer blows or older revocations fall off the list, thus reinstantiating those revoked keys. Using a certifying-authority model avoids these vulnerabilities.

TOKENS AND PASSWORDS

Secure systems may use a mix of ways to verify the identity of a user: The verification can be from something you have, such as a token, smart card, or PCMCIA card; something you are, using biometrics for fingerprint, voice, face, or other forms of recognition; or something you know, such as a password.

The "poster child" for tokens and secure key packaging is the smart card. A cryptoengine inside the smart card is isolated from the rest of the system, and attackers cannot easily compromise it. Because the processor and memory reside on the same chip, the key never travels over an external bus that is vulnerable to observation. You don't have to use smart-card chips in a smart card. A set-top box, for example, could use a smart-card chip to implement secure key generation, storage, and limited decryption.

Tokens should be impossible to forge and should be attached to one user to prevent their use when stolen. Biometric methods tie a token to a user on a physical level (fingerprint), and passwords do so based on a user's knowledge. However, passwords without tokens are insecure; after all, a computer can crack any password that a person can remember.

TABLE 1—POTENTIAL APPLICATIONS REQUIRING ENCRYPTION

Application	Attack	Standard	Role of encryption
Virtual private network (VPN)	System exposed; insecure Internet used as backbone for secure intranet	IPSEC	Creates secure data channel, transparent to applications
Bridges, for example, between applications and the lower, more vulnerable levels of the network	Snooper application accessing data from other applications	Secure-sockets-layer (SSL) protocol	Protects system from unintended access
Bar-top computer game	System exposed; wireless capture of data	DES for transfer to credit-card companies	Protects credit-card numbers sent wirelessly to back-room server for processing
Set-top box or Internet appliance	Pirate box	Various	Controls Internet/interactive services or prepaid premium viewing
Digital security camera	Must physically compromise internal network; for example, wire splicing	Closed networks; no standards	Prevents "nothing's happening" signal from being spoofed over real signal

Also, a strong key protected solely by a password is only as secure as the weaker password.

Note that smart cards are vulnerable; attacks based on differential power analysis can provide attackers with information on the internal state of the smart card (**Reference 3**). Proposed changes to prevent attackers from "reading" smart cards or other ICs through such information leaks include current scrambling, which breaks up current patterns by inserting extra and random wait states so that patterns are more difficult to catch, and reducing variances in internal currents so that internal states are more difficult to determine.

DAMAGE CONTROL

Despite all the precautions you can take to prevent the compromise of your encryption system, you should assume that someone can and will break your encryption scheme. If an attacker discovers a backdoor, flaw, or vulnerability, you need to be able to recover and correct it.

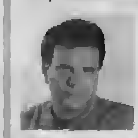
The first step is determining whether someone has breached your system. If the attacker's motive is to damage or crash your system, you'll probably quickly find out this fact. However, if the attacker wants information only to use it elsewhere, such as with credit-card numbers, or to abuse a function, such as adding value to a smart card, no fireworks point out these attacks. Filters can uncover this second kind of breach by observing system behavior and watching for aberrations, such as a smart-card account making an unusual number of transactions.

Another method for detecting a breach is to keep a log of every transaction. In a set-top environment, only the home office should ever communicate with a node box. Therefore, if the transaction logs fail to match, then someone has attempted to contact the node, signaling that someone may have compromised the node. The more information you can capture, the better your chances of tracking and understanding a breach. The problem arises, however, that an attacker can use any debugging hooks you include in the final product to compromise it.

WHEN IS "GOOD ENOUGH" GOOD ENOUGH?

The foundation of a good security scheme is that the cost to steal information should be more than the information is worth. Remember also that encryption works in parallel with physical protection: An attacker needs to break both to steal information. If someone has access to your encrypted data, then the attacker has already compromised your system. Encryption is managing the compromise by placing another barrier in the path of an attacker before the attacker can abuse the information. Some systems have built-in physical protection; set-top boxes connect to conditional-access lines that already protect content. A VPN, however, uses access lines and the Internet, which offer little or no protection. Encryption is about trust and preventing access to those without trust. Security offers the best results when employing a combination of countermeasures.

You can reach Technical Editor Nicholas Cravotta at 1-510-558-8906, fax 1-510-558-8914, cravotta@compuserve.com.



Encryption may seem a trifle overdone. Cryptography tends to predict the worst to prevent surprises. Secure encryption, therefore, must balance protection, access, performance, risk, and cost in the pursuit of "sufficient security." One reliable metric is that the more complicated your implementation, the more vulnerable it is. Aim for simplicity, and the rest should follow. □

REFERENCES

1. "Cracking DES," Electronic Frontier Foundation, O'Reilly and Associates, Sebastopol, CA, 1998, www.eff.org.

2. Advanced Encryption Standard Development Effort, www.nist.gov/aes.

3. Kocher, Paul, Joshua Jaffe, and Benjamin Jun, "Introduction to Differential Power Analysis and Related Attacks," www.cryptography.com/dpa.

4. Waltz, Edward, *Information Warfare: Principles and Operations*, Artech House, Boston, MA, 1998.

5. Stallings, William, "Encryption Choices Beyond DES," *Communication Systems Design*, November 1998, pg 37 to 43.

6. 1999 RSA Data Security Conference Proceedings, 1-415-544-9300.

7. Schneier, Bruce, "Security Pitfalls in Cryptography," www.counterpane.com/pitfalls.html.

8. Schneier, Bruce, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Wiley & Sons, 1996.